EPFL Machine Learning CS-433 - Project 2 report

Edoardo Tarek Hölzl and Hugo Vincent Roussel
Department of Computer Science, EFPL Lausanne, Switzerland

***Abstract*—Detecting roads from satellite imagery is of great interest for a variety of applications such as paving the way for automatic road navigation or assessing infrastructure damages after natural disasters. We describe the steps we followed to train a convolutional neural network that classifies pixels of an image into road or background and getting an accuracy of 90.9%.**

I. Introduction

There are currently more than 5000 satellites orbiting the globe today providing us with ever greater quantity of high definition images of the Earth. This data influx is not expected to slow down, with the costs of putting into orbit man made satellites getting ever lower thanks to the recent incursion of the private sector in this industry, which was previously reserved to states. Treating this data in an efficient manner is critical and can be done with a class of machine learning techniques called Convolutional Neural Networks (CNNs) that show remarkable performance in general classification and recognition tasks. The objective of this project is to classify patches of pixels of satellite images into two categories, road and non-road pixels. In this paper, we will describe the steps we took in order to build and train a CNN model that generates probabilities for each pixel.

II. Data-set

The provided data set consists of 100 images with their respective masks indicating the location of road pixels, and a set of 50 test images. Training images have a resolution of 400x400 pixels, with 3 color channels (RGB), while test images have a higher resolution of 608x608, with the same number of channels. The competition evaluation metric is chosen to be the F1-score, and is not measured on a pixel level, but per predicted patch of 16x16 pixels. The F1-score is a metric for binary classification that combines both precision and recall of a classifier using a harmonic mean, and is preferred over classical accuracy when both False Positives and False Negatives are crucial at assessing the model's performance. Since this data-set consists of a majority of negative examples, F1-score is preferred over accuracy. We did not perform any kind of data cleaning.

III. Methods and models

A. Baseline Model

We are provided with a baseline model with the project statement, and used it as our baseline to assess the minimal performance we should expect from more complex models.

This model performs classification over patches of 16x16, and it's architecture is similar to a vanilla image classifier: Two back-to-back 2D convolutions with a filter size of 5x5, each followed by a ReLU activation function and a MaxPooling layer. The output of convolutions is fed into a fully-connected layer with 512 neurons, before finally activating the last fully-connected layer that has 2 neurons (one for each class).

This model is trained with a Cross-Entropy loss and balances out the classes. The model yields an accuracy of 79.04% on the test set, it can probably be improved with additional tuning. The problem with this model is his lack of awareness of the context from neighboring regions, it has however the advantage of being trained on the competition's evaluation metric (classification of patches).

B. U-Net

The U-Net was developed in 2015 [1] and was initially designed for Biomedical image segmentation . The architecture consists of an Encoder and a Decoder path, with only Convolutional layers and no fully-connected layers. Images fed into this network are first down-sampled in multiple steps (contracting path) and then up-sampled (expansive path) using combinations of 2D Convolutions, MaxPooling layers and Transposed Convolutions. The architecture as described in the paper is depicted in Fig. 1 below. We have used a slightly modified U-Net, that has the contracting and expansive paths, but we added intermediary layers that give us better performance. The architecture of our modified U-Net implementation is described in the next section.

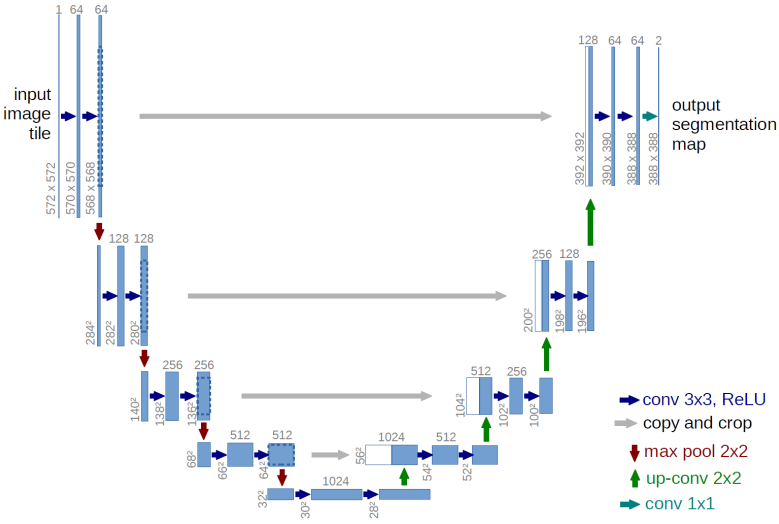

Fig. 1: U-Net as described in the original paper

IV. U-Net architecture implementation

The original paper describes the general encoder and decoder aspect of this model. However, all other parameters, such as the number of filters, filter sizes, depth, padding and activations are very important as they could increase the performance. In this part, we describe the different values we have tried, and the results associated to them.

1) Filter Size: We have tried filter sizes of 3x3 and 5x5, at all convolutional units, but found that they do not affect performance in a significant manner. We resorted to using the simplest 3x3 kernel size for all convolutions.

2) Padding: The paper specifically describes un-padded convolutions, so the borders of the input image were not predicted on. This could be a problem, as we would like to have border detection. One way to fix this, would be to pad the original image with a certain number of pixels, that would be cropped by the network, or to use a padding of 1 in every convolutional unit we use. Both methods were assessed and produced similar results. Expanding the original image made training slower by a factor of almost 2, so we went with the second solution, and used padded convolutions.

3) Model Depth: The depth of the model refers to the number of MaxPooling layers in the contracting path. We started with a depth of 4, and gradually reduced the depth, while monitoring validation performance and training time. A higher depth increases the training time, as more parameters have to be estimated, but does not necessarily increases performance. Also, the depth determines the image resolution at the deepest layer. Each MaxPooling layer divides the resolution by 2 (window of size 2 is used), so with an input image of size WxW and a depth of d, the resolution at the deepest layer is :

$$\frac{W}{2^d} \tag{1}$$

The result of this fraction must be an integer, as if the image size is not divisible by this factor, then MaxPooling will not work as expected. The chosen depth was of 3, as it reduced training time and did not affect performance.

4) Number of filters: The original architecture doubles the number of filters at every layer of the contracting path, and halves them in the expanding path. So the only choice is the number of filters at the first layer. We experimented with 16, 32 and 64 filters. The results are depicted in table I below

Depth	# Filters at 1^{st}	Patch Size	Step	LR	Batch Size	Epochs	F1-Score
3	16	80	20	0.001	128	18	0.847
3	32	80	20	0.001	128	18	0.852
3	64	80	20	0.001	128	27	0.867

TABLE I: Results for different Filter sizes (without BN and Dropout), trained with BCE

5) Activation: As of today, ReLU is the go-to choice activation function. It is used in the original paper after every convolution. We also use Leaky-ReLU [2], which is a variant of ReLU and is defined below, with alpha usually in the orders of 10^{-1} :

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha * x & otherwise \end{cases} \tag{2}$$

6) Batch Normalization & Dropout: In the U-Net paper [1] dropout layers are mentioned, as they would induce further data augmentation. They recommend using them in the last layers of the contracting path, which is what we did. We also added Batch Normalization at every layer of the contracting path, after each convolution. Using those two layers significantly decrease the convergence time. Combining them provides regularization, prevents over-fitting [3] and increases performance.

V. Data Augmentation

The small size of the initial data set is not enough for training a CNN. By using the original U-Net architecture, and training on the 100 images at full resolution, we saw that it easily over-fits on the training data and obtains a validation F1-score close to 98%, but performed poorly on the test set. This might be due to the fact that most roads are straight and parallel. To avoid over-fitting, we used different image augmentation techniques that multiplied the number of training images by a certain factor.

A. Patching

To increase the number of training data points, we generated multiple patches from the original images. Patching has been proved to increase performance [4], and helps us increase the data-set size. However, patches should not be too small, (loss of context), but also not too big. As mentioned, the input image size is constrained by the model depth (Section IV.3), so we would need a patch size that is divisible by 2^d where d is the model depth.

Since the chosen depth is 3, and we did not want to have too small patches, we chose a patch size of 80x80. We create patches using a sliding window of 20 (an overlap of 60 pixels), resulting in $17^2 = 289$ patches per image. This results in a training set of 28900 images.

B. Rotations

Convolutional layers are quite sensible to rotations. Intuitively, by feeding the network with straight roads, it does not learn the possibility of different angles. Hence, we decided to further augment images using rotations of different angles. We chose all angles that are multiples of $15°$ from 0 up to $75°$, and rotated the full size image to all of those degrees.

This multiplies the training size by 5. Rotating increases the image size, and leaves out black patches at the corners. To fix this, we mirrored the image with a certain padding and cropped the rotated image to a size of 400x400. Furthermore, we also added stochastic image augmentation for rotations of

$x * 90°$, and horizontal/vertical flip.

Those described augmentation techniques multiply the training size by a factor of 5x289 = 1445, so we have $144,500$ training images.

VI. Training & Prediction

Training convolutional networks needs quite a bit of computing power in memory. Not having a GPU at hand would require huge training times, so we rented an instance on GCP, with a GPU Nvidia Tesla K80 (12GB of memory). All models were optimized using Adam optimizer with the same learning rate, but we tried using 3 different loss functions for the final model: Binary Cross-Entropy, BCE with class balancing, Dice Loss.

A. Training

1) Without Rotations: We first trained our models using the patching described, but without any Dropout or Batch Normalization layers, and without rotation or stochastic image augmentation, optimizing the regular BCE loss. The results for different number of filters in the 1^{st} layer are described in table I. Performance was clearly increase compared to the baseline model, and using 64 filters at the first layer also increases the F1-score. Hence, we decided to train all subsequent models with 64 filters.

2) BCE vs BCE with class balance: Since the data-set contains more negative examples than positive, we tried using class balance to increase recall. This technique does in fact increase recall, while keeping the precision at the same level. The weight of positive examples is increased by a factor num_neg/num_pos per batch. The results of those experiments can be found in table II.

Loss	LR	Batch Size	Epochs	F1-Score
BCE	0.001	128	27	0.867
BCE with Balance	0.001	128	27	0.872

TABLE II: Results for different loss functions. Patch=80, step=20, Filters=64, Depth=3, no Dropout, no BN, no augmentation

3) Dice Loss: At our final iteration of training models, we tried replacing the BCE loss with the Dice Loss. This type of loss can be seen as a differentiable, continuous F1-score. It seemed right to use the closest loss function to the evaluation metric, as it would be optimized directly.

B. Prediction

Prediction can be done directly on the 606x608 test images, as $\frac{608}{2^3} = 76$. In order to increase the score, we also added ensemble prediction, where each image is predicted 4 times, with rotation angles [0, 90, 180, 270].

Also, we noticed that combining models that were trained with different loss functions give better results. Using Dice loss makes the model have a high precision, but lower recall, while using BCE with balance loss, the precision is lower and recall is higher. Those differences can be clearly seen by looking at the predictions of each model in Fig. 2. Hence, by combining the output of both models, we can increase the F1-score.

Each 16x16 patch is predicted based on the mean value of the patch in the segmented image. If the mean is ≥ 0.5 then it is a road, otherwise background.

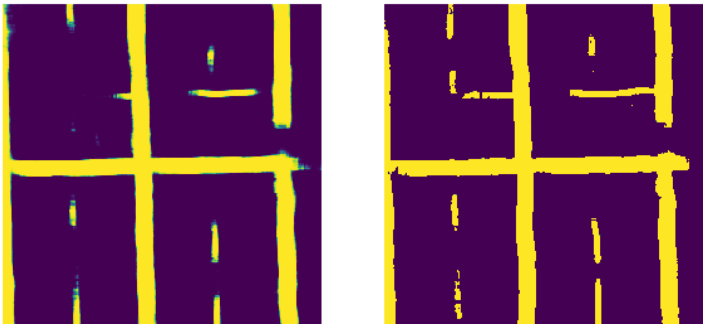

Fig. 2: Predictions using BCE with balance (left) and Dice loss (right)

C. Implementation Details

The model was implemented using the PyTorch framework for Deep-Learning, and can be found on `github.com/ehoelzl/road_segmentation`. We used TensorBoard to monitor the progression of the training of the model, as well as observe example predictions on a validation set. BCE loss was used directly as provided in the framework, but we added the Dice Loss ourselves, as it is not available.

VII. Results

In this section, we describe our final obtained results using all the techniques described above. We first predicted using each model independently, without ensemble prediction, in order to assess it's absolute performance on a validation set, with augmented images. The best model was when we used Dice loss with ReLU activation function with Batch normalization and dropout, and ensemble prediction. The model was trained on patching technique described above, as well as rotations in various angles, for around 12 hours. The final F1-score on the Kaggle competition was 90.9%, placing us 12th in ranking.

We believe to have delivered a pretty solid model, that is quite powerful and provides solid results. It was nice to see the different effect of loss functions on the output of the model, as well as on the different metrics we used.

Loss	Prediction Rotation	Activation	Epochs	Time	F1-Score
BCE with Balance	FALSE	ReLU	30	14h	0.902
Dice Loss	FALSE	ReLU	27	12h	0.90
BCE with Balance	TRUE	ReLU	30	14h	0.906
Dice Loss	TRUE	ReLU	27	12h	0.909
Dice Loss	FALSE	Leaky-ReLU	17	6h30	0.903
Dice Loss	TRUE	Leaky-ReLU	17	6h30	0.908

TABLE III: Final results. LR=0.001, Patch=80, step=20, Filters=64, Depth=3, Dropout=0.2, BN, with augmentation

VIII. Conclusion

This project provided us with a real world problem to work on. We were able to study different CNNs architectures and build and train our own implementation of an U-Net. To then augment our F1-Score we combined multiple models, experimented with the loss functions and augmented the original dataset through patching and rotations. We are sure that using variations of the proposed architecture and optimizing further the hyper parameters one could get an even better score. We experimented with combinations of two models' outputs and taking the mean of both outputs. This method gives a slightly better score, but multiplies prediction time by 2 (around 40 seconds per image), so we dropped it in favour of faster prediction.

References

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation* arxiv.org/abs/1505.04597

[2] *Empirical Evaluation of Rectified Activations in Convolution Network* arxiv.org/pdf/1505.00853.pdf

[3] Chang-Yu Hsieh, Benben Liao, Shengyu Zhang, Guangyong Chen, Pengfei Chen, Yujun Shi. *Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks* arxiv.org/pdf/1905.05928.pdf

[4] *Patch-based Convolutional Neural Network for Whole Slide Tissue Image Classification.* www.ncbi.nlm.nih.gov/pubmed/27795661